SCALABLE SERVER ARCHITECTURE BASED ON ASYMMETRIC 3-WAY TCP

This application is a Continuation-In-Part of application Ser. No. 09/650,644 field on August 30, 2000.

FIELD OF THE INVENTION

[0001] This invention relates to client-server network processing. More specifically, this invention relates to a server architecture based on asymmetric 3-way TCP.

BACKGROUND OF THE INVENTION

[0002] Computers on the Internet communicate with each other by sending packets according to a standard protocol suite including the TCP/IP (Transmission Control Protocol and Internet Protocol) suite.

[0003] As the usage of Internet increases, high-performance web servers or mail servers are desired that can accommodate ever-increasing service requests from many clients. Especially, what is desired is a scalable server system whose handling capacity can increase in proportion to the frequency of service requests from clients.

[0004] One way to implement such a scalable server system is to employ a high-performance single-processor server. But, this is not an economically desirable solution because one must pay a stiff price for such a single-processor high-performance system. Another way is to employ multiple servers to distribute service requests from clients. However, adding servers does not necessarily translate to increased performance because the existing protocol limits handling of service requests to only those servers specified by the clients, thereby creating a bottleneck.

[0005] Therefore, there is a need for a scalable server architecture whose performance is proportional to the client's demand.

[0006] U.S. Pat. No. 5,774,660, issued to Brendel et al. on Jun. 30, 1998, discloses a load balancer operating in a TCP/IP environment. As shown in Figs. 10, 11A, 11B, 12, and 17 of U.S. Pat. No. 5,774,660, the load balancer and a server establish a TCP playback, i.e., TCP/connection, before the load balancer relays a packet received from a client to the server. In order to establish such a connection, each of the load balancer and the server should have a TCP/IP stack. As described above, the load balancer (i.e., a front-end server) and the server (i.e., a back-end server) establish a TCP connection so that the TCP connection between a browser (i.e., the client) and the load balancer is transferred to the TCP connection between the browser and the server. The load balancer then transfers the TCP/IP connection and a current TCP state to an assigned server (i.e., the back-end server), using TCP state migration. TCP state migration is not simply forwarding packets through as they are received. Instead the packets received are stored by the load balancer and then played back to the assigned server.

[0007] However, a service request, which is received by the load balancer, is not processed by the assigned server by sending the service request to the assigned server in a data link frame such as an Ethernet frame, without establishing a TCP/IP connection between the front-end server and the back-end server.

[0008] Therefore, there is a need for a scalable server architecture whose performance is performed without establishing a TCP/IP connection between the front-end server and the back-end server.

SUMMARY OF THE INVENTION

[0009] An object of the present invention is to provide a scalable server system whose performance does not degrade with the increased frequency of the client service requests.

[0010] Another object of the present invention is to provide a server system that can distribute service requests from a client over multiple processors to increase the performance.

[0011] In accordance with one aspect of the present invention, there is provided a server system for processing a service request from a client in a network processing environment, which comprises a front-end server and at least one back-end server connected through a data link. The front-end server receives a service request from the client and generates a data link containing a service command necessary to perform the service request. The back-end server connected to the front-end server through the data link receives the data link frame containing the service command and executes the service command and sends directly the result of the service command to the client in a pseudo packet bypassing the front-end server. The client then recognizes the pseudo packet bearing the result of the service command as a packet originating from the front-end server.

[0012] In accordance with another aspect of the present invention, there is provided a method for processing a service request from a client in a network processing environment. The method comprises the steps of receiving a service request from the client by a front-end server, processing the service request by a back-end server connected to the front-end server through a data link, without establishing a TCP/IP connection, and providing the processed result to the client directly by the back-end server in a pseudo TCP packet bypassing the front-end server.

[0013] In accordance with still another aspect of the present invention, there is provided a method for processing a service request from a client in a network processing environment. The method comprises the steps of receiving a service request from the client by a front-end server, processing the service request by a back-end server connected to the front-end server through a data link, without establishing a TCP/IP connection, and providing the processed result to the client directly by the back-end server in a pseudo

UDP packet bypassing the front-end server.

[0014] In accordance with yet another aspect of the present invention, there is provided a computer-readable medium containing a computer program. The computer program comprises the steps of receiving a service request from the client by a front-end server, processing the service request by a back-end server connected to the front-end server through a data link without establishing a TCP/IP connection, and providing the result to the client by the back-end server bypassing the front-end server.

[0015] The foregoing and other objects and aspects are accomplished by providing an asymmetric 3-way TCP (ATCP) scheme that includes a front-end server and a back-end server, both communicating with a client. The front-end server receives a service request from the client and sends a service command contained in a data link frame necessary to process the service request to the back-end server through a data link. The back-end server executes the service command contained in the data link frame and sends the result back directly to the client in a pseudo TCP packet which would be recognized by the client as being sent by the front-end server. The pseudo TCP packet has the IP address and the port number of the front-end server to look like a packet sent by the front-end server.

BRIEF DESCRIPTION OF THE DRAWINGS

[0016] The above and other objects and features of the present invention will become apparent from the following description of preferred embodiments given in conjunction with the accompanying drawings.

Fig. 1 is an illustration of an asymmetric 3-way TCP (ATCP) scheme of the present invention.

Fig. 2 is a schematic diagram of the ATCP of the present invention.

Fig. 3 is a flowchart of a software routine run at the front-end server.

Fig. 4 is a flowchart of a software routine run at the back-end server.

Fig. 5 is a flowchart of a software routine run by the front-end server to send a service command the back-end server through an ATCP_command_back-end_server packet.

Fig. 6 is a flowchart of a software routine run at the back-end server upon receiving the ATCP_command_to_back-end_server packet from the front-end server.

Fig. 7 is a flowchart of a software routine run at the back-end server before sending an ATCP_service_reply packet to the client and an ATCP_ACK packet to the front-end server.

Fig. 8 is a flowchart of a software routine run at the front-end server upon receiving the ATCP_ACK packet from the back-end server.

Fig. 9 is a flowchart of a software routine run at the front-end server upon receiving an ACK packet from the client.

Fig. 10 is a flowchart of a software routine run at the back-end server upon receiving the ATCP_ACK packet from the front-end server.

Fig. 11 is a block diagram for a back-end server that assembles a pseudo TCP packet.

Fig. 12 is a state transition diagram of a state machine controlling the back-end server.

DETAILED DESCRIPTION OF THE PRESENT INVENTION

[0017] Fig. 1 illustrates the operations of a client-server system based on an asymmetric TCP (ATCP) of the present invention. The system has a client 11 communicating with a front-end server 12 and a back-end server 13. The operations may be summarized as follows:

[0018] (1) The client makes a service request to the front-end server through the socket

of the front-end server whose TCP connection is specified by the client. Note that the client does not specify any connection with the back-end server and does not even know the presence of the back-end server.

[0019] (2) The front-end server relays the service request received from the client to the back-end servers through a data link together with the source/destination port numbers and source/destination IP addresses and the acknowledge number expected by the client and the sequence number of the TCP packet that would contain the result of the service request. The front-end server may simply relay the client's request to the back-end server or sends a specific command after decoding a request and generating a data link frame containing the specific command. If the front-end server directly handles the client's service request, the front-end server would send the result to the clients through a conventional TCP packet.

[0020] (3) The back-end server, which is connected to the front-end server through the data link, without establishing a TCP/IP connection, sends a TCP packet containing the service result to the client through the client's socket. Although the TCP packet is made by the back-end server, the packet is disguised as if it were sent by the front-end server by using the destination/source sockets specified by the client and the front-end server. For this reason, the TCP packet sent by the back-end server is referred to as a "pseudo TCP packet."

[0021] (4) The client sends a TCP packet responding to the service received to the front-end server and sends a further service request to the front-end server, if necessary.

[0022] Fig. 2 shows a schematic diagram of the TCP and ATCP packets exchanged between the client, the front-end server, and the back-end server.

[0023] (a) SYN 21, SYN ACK 22, ACK 23 are conventional TCP packets for handshaking between the client and the front-end server to establish a TCP connection. PSH 24 is a conventional TCP packet containing a service request from the client to the

front-end server.

[0024] (b) PSH' 24' is a conventional TCP packet that would have been generated by the front-end server to send the front-end server's reply to the service request. However, in the present invention, PSH' 24' is not generated. Instead, the front-end server relays the service request to the back-end server so that the reply to the service requests is handled by the back-end server. By using multiple back-end servers, the front-end server can distribute the processing among multiple processors, thereby enabling a high-performance server system.

[0025] (c) ATCP_command_to_back-end_server 25 of the present invention is not a conventional TCP packet, but an ATCP packet containing a service command sent from the front-end server to the back-end server through the data link, the source/destination IP addresses of the client requesting the service, the port number of the client, the acknowledgement number expected by the client, and the sequence number expected by the client.

[0026] (d) The back-end server, receiving a service command from the front-end server through ATCP_command_to_back-end_server 25, processes the command and sends the result directly to the client in ATCP_service_reply 26. For ATCP_service_reply 26 to be recognized by the client process through the client's TCP layer, the ATCP_service_reply packet must look as if it were sent by the front-end server. This is accomplished by making the destination port number of the ATCP packet 26 assigned the client's port number and the source port number of the ATCP packet assigned the front-end server's port number. The destination port number and the acknowledgement number of the ATCP_service_reply packet 26 are assigned the sequence number and the acknowledgement number received through the ATCP_command_to_back-end_server 25 packet.

[0027] (e) After sending ATCP_service_reply 26 to the client, the back-end server sends

an acknowledgement to the front-end server through ATCP_ACK 27. Without this acknowledgement, the front-end server has no way of knowing whether the back-end server correctly has processed the command.

[0028] (f) The TCP layer of the front-end server sets the acknowledgement number expected from the client as the sequence number sent to the back-end server plus the size of data sent to the client by the back-end server.

[0029] (g) Receiving ATCP_service_reply 26, the client recognizes it as a packet received by the front-end server and sends an acknowledgement ACK 28, a conventional TCP packet containing an acknowledgement number, to the front-end server. The acknowledgement number would be equal or less than the acknowledgement number expected by the front-end server because the front-end server already updated the expected acknowledgement number in step (f). If the expected acknowledgement number had not been updated in step (f), a TCP packet having the response larger than the expected number would have been discarded.

[0030] (h) Receiving ACK 28, the ATCP of layer of the front-end server sends a response ATCP_ACK 29 if the acknowledgment number received is equal or less than the expected acknowledgement number.

[0031] (i) Receiving ATCP_ACK 29, the ATCP layer of the back-end server empties the buffer by the received amount. Since the front-end server and the back-end server of present invention cooperate each other with respect to the acknowledgement number without disturbing other TCP services, ATCP_ACK 29 is treated similar to a TCP packet.

[0032] The front-end server relays the TCP packet received from the client through the established connection between the client and the front-end server. Note, however, that the back-end server need not and does not establish a TCP connection with the client because the back-end server sends the result in a pseudo TCP packet that would look like a TCP packet sent from the front-end server. In fact, the client does not even know of

the presence of the back-end servers.

[0033] The packets exchanged by the front-end server and back-end server need not be a TCP packet or of the same format as a TCP packet as long as the packets support a connection-oriented service.

[0034] Fig. 3 is a flowchart of a software routine run at the front-end server. The process at the front-end server first initializes the socket connection with the back-end server (step 101), and blocks itself while waiting a service request from a client (step 102). When a service request is received from a client, the process at the front-end server is awakened, and the process delivers a service command contained in a data link frame to the back-end server through the ATCP layer. The process at the front-end server repeats steps 102-104. Processing point (A) represents the point in time when the process run at the front-end server sends a command to the back-end server through the ATCP layer.

[0035] Fig. 4 is a flowchart of a software routine run at the back-end server. The process run at the back-end server first initializes the socket connection (step 112) and blocks itself while waiting for a command from the front-end server (step 113). The process is awakened when ATCP_command_to-back-end_server 25, a pseudo TCP packet containing a command from the front-end server, is sent to the ATCP level of the back-end server (step 114).

[0036] Please note that there is no need for a TCP stack at the back-end server because the back-end server receives a special command from the front-end server rather than a full TCP packet.

[0037] After executing the command received from the front-end server (step 115), the process incorporates the service result into a pseudo TCP packet and send it to the client, not to the front-end server, and blocks itself waiting for a next command. The process at the back-end server repeats the above steps (steps 113-116). Processing point (C) represents the point in time when the command is delivered from the back-end server

software to the ATCP level at the back-end server through the API such as a socket.

[0038] Fig. 5 is a flowchart of a software routine run at the front-end server after step (A). The process run at the front-end server obtains the source/destination IP addresses of the client requesting the service, the source/destination port numbers, the acknowledgement number, and the sequence number (step 105), then generates an ATCP packet (step 106), and sends it to the back-end server (step 107).

[0039] Fig. 6 is a flowchart of a software routine run at the back-end server. The back-end server runs a back-end server command process for executing the command received from the front-end server through the data link (step 108). The software at the back-end server has an ATCP layer that sets up a 3-way ATCP connection (step 109) and wakes up the back-end server command process, which was blocked while waiting for a command from the front-end server (step 110).

[0040] Fig. 7 is a flowchart of a software routine run at the back-end server before sending ATCP_service 26 to the client and ATCP_ACK 27 to the front-end server (step 119). When the result of the service command is received from the back-end server command process (step 117), it is sent to the client by incorporating the IP address and port number of the front-end server (step 118).

[0041] Fig. 8 is a flowchart of a software routine run at the front-end server upon receiving ATCP_ACK 27 from the back-end server at the time indicated by Point D. The front-end server checks whether the ATCP connection is valid (step 120). If not valid, the received packet is discarded (step 121). If valid, the front-end server extracts the size of data transmitted by the back-end server to the client (step 122) and updates the expected acknowledgement by adding the data size (step 123).

[0042] Fig. 9 is a flowchart of a software routine run at the front-end server upon receiving ACK 28 from the client at the time indicated by Point E. The front-end server checks whether the ATCP connection is valid (step 124). If not valid, the received

packet is discarded (step 126). If valid, whether the received ACK number is equal or less than the sequence number is checked (step 125). If the received ACK number is larger, the received packet is discarded (step 126). If the received ACK number is equal or less, the ATCP buffer is emptied by the received amount (step 127).

[0043] Fig. 10 is a flowchart of a software routine run at the back-end server at receiving ATCP_ACK 29 that was relayed by the front-end server receiving ACK 28 from the client. Point F in FIG. 10 corresponds to point F in Fig. 9, the time when the back-end server received ATCP_ACK 29 from the front-end server.

[0044] Receiving the ATCP_ACK packet from the front-end server (step 128), the back-end server checks whether the ATCP connection is valid (step 129). If not valid, the received packet is discarded (step 131). If valid, whether the received ACK number is equal or less than the sequence number is checked (step 130). If the received ACK number is larger, the received packet is discarded (step 131). If the received ACK number is equal or less, the ATCP buffer is emptied by the received amount (step 132).

[0045] Fig. 11 shows the back-end server which comprises a back-end server controller 30, an NIC(network interface card) 40, a disk 50, a CD 60 and memory 70. The NIC functions as an interface between the back-end server and the network cabling.

[0046] The back-end server controller 30 is preferably a program controlling various controlling operations in the back-end server. The back-end server controller 30 further comprises a state machine 31 for controlling the back-end server, a NIC controller 32, a command decoder 33, an ALU 34, a disk controller 35, a CD controller 36, and a pseudo TCP packetizer 37.

[0047] The NIC controller 32 is a program module for controlling receipt and transmission of packets from and to the network by reading and writing an NIC register. The command decoder 33 is a program module to decode the command from the front-end server included in the TCP packet received through the NIC. The ALU 34 performs

arithmetic and logic operations necessary for performing a command. The disk controller 35 is a program module to control the input and output to and from the disk 50. The CD controller 36 is a program module to control the output from a CD from the CD 60. The pseudo TCP packetizer 37 assembles a pseudo TCP packet by making up the TCP header and the IP header to look like that the packet was sent by the front-end server.

[0048] Fig. 12 is a state transition diagram of a state machine controlling the back-end server of the present invention. The back-end server is ready to receive a command from the front-end server through the NIC. When a command is received, the received command is decoded by the command decoder 33 to execute the command or to perform I/O. The result is incorporated by a pseudo TCP packetizer 37 into a pseudo TCIP/IP packet that looks like a packet sent by the front-end server. The above process is repeated.

[0049] The asymmetric 3-way TCP (ATCP) scheme of the present invention realizes an economic and scalable server system where the server distributes a client request to one or more back-end servers which perform the work and deliver the result to the client.

[0050] The ATCP scheme of the present invention can increase the performance of a web server by replacing the existing TCP layer with the ATCP layer so as to distribute the work among multiple back-end servers. For example, the existing web servers may be modified in such a way that the web servers that received the requests from clients may instructs the multiple back-end servers to perform the work and relate the work to the clients.

[0051] The presence of back-end servers is transparent to the client in that there is no need for a client to have any prior knowledge of back-end servers and there is no need to change the IP or the port address of the existing servers. Only the front-end server needs to have prior knowledge of the back-end servers.

[0052] Although the prevent invention was described in terms of a preferred

embodiment using the TCP, an alternative embodiment is possible using the UDP (User Datagram Protocol), another transport-layer protocol offered as part of the TCP/IP suite. Whereas the TCP provides a connection-oriented service where a connection is established first before packets are sent in a continuous stream, the UDP provides a connectionless service where individual datagrams are sent without first establishing a connection. The UDP is typically used where guaranteed delivery is not critical, such as video or audio transmission.

[0053] In the alternative embodiment, similar to the pseudo TCP packet, the front-end servers sends to the back-end server the UDP header information (the front-end server's UDP port number and the client's UDP port number) and the IP header information (the front-end server's IP address and the client's IP address). The back-end server then assembles a pseudo UDP/IP packet using the information received.

[0054] While the invention has been described with reference to preferred embodiments, it is not intended to be limited to those embodiments. It will be appreciated by those of ordinary skilled in the art that many modifications can be made to the structure and form of the described embodiments without departing from the spirit and scope of this invention.

[0055] For example, the preferred embodiment of the present invention was described with respect to a web server, but those skilled in the art will appreciate that the same technique can be applied to other kinds of servers such as a DB server, a file server, a mail server, a firewall server and a printer server.